Message Logging in Mobile Computing

Bin Yao[†], Kuo-Feng Ssu[‡], and W. Kent Fuchs[†]

[†] School of Electrical and Computer Engineering, Purdue University
[‡] Coordinated Science Laboratory, University of Illinois

Abstract

Dependable mobile computing is enhanced by independent recovery, low power consumption and no dependence on stable storage at the mobile host. Existing recovery protocols proposed for mobile environments typically create consistent global checkpoints that do not guarantee independent recovery and low power consumption. This paper demonstrates the advantages of message logging by describing a receiver based logging protocol. Checkpointing is utilized to limit log size and recovery latency. We compare the performance of our approach with that of existing mobile checkpointing and recovery algorithms in terms of failure free overhead and recovery time. We also describe a stable storage management scheme for mobile support stations. Garbage collection is achieved without direct participation of mobile hosts.

1 Introduction

Mobile computing [1,2] presents new challenges and requirements for checkpointing and recovery protocols [3]. Failures such as loss of connection or power outages that are rare in fixed networks can be common in mobile environments. Recovery algorithms are required to tolerate multiple simultaneous failures and failure during recovery, and it is desirable that processes be able to recover independently. Coordinated recovery among processes running on Mobile Hosts (MH) may slow down recovery [4] and increase the chance of having multiple rollbacks of the entire system in order to handle errors during recovery. Conserving battery power by means of limiting the number of extra messages during checkpointing and recovery is also important. Limiting additional transmitted messages has the added benefit of reducing contention on the wireless network.

As an MH may be lost or permanently damaged, hard drives on mobile hosts are not generally considered stable storage. Therefore they are not suitable as the only location for storing checkpoints or message logs. Traditional checkpointing and message logging algorithms [5–12] are not directly applicable under such conditions. Previous proposals have suggested that checkpoints be sent back to Home Agents (HA) [13]. Others have proposed that stable storage on Mobile Support Stations (MSS) be used for checkpoints and message logs [14–16]. Because checkpoints and/or message logs are stored on different MSSs as an MH moves from cell to cell, the organization of the distributed process state information is important for successful recovery. Several algorithms have been proposed to solve these problems [14, 17]. Garbage collection of stable storage on MSS is also of importance. When state information on MSS is no longer needed for recovery, it should be discarded to make room for new checkpoints and message logs. Most present schemes require cooperation from mobile hosts. If one participating MH fails, some stable storage may never be collected for reuse.

Checkpointing and recovery protocols previously proposed for mobile environments have typically saved consistent global checkpoints [13, 15, 18, 19]. This requires all participating mobile hosts to roll back during recovery, but some mobile hosts may not be able to rollback due to transient or permanent failures. This approach forces application messages to be resent over the slow wireless network during recovery, resulting in slow recovery and additional power usage at the MH. Recent work has shown through analytic modeling that message logging can be an attractive approach to recovery in mobile environments [14]. Another recent research project has derived mechanisms for managing stable storage on the MSS [20].

This paper describes a receiver-based pessimistic message logging protocol for MH, MSS and HA, and a distributed state organization scheme for mobile computing environments. Using our approach, processes running on mobile hosts are able to recover quickly and independently of other processes. The protocol is experimentally compared with an ideal consistent checkpoint protocol to demonstrate that message logging incurs similar failure free overhead and achieves much faster recovery in a wireless network implementation. This approach requires no extra control messages sent by the MH. Garbage collection

This research was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract DABT 63-96-C-0069, and in part by the Office of Naval Research under contract N00014-97-1-1013.

is achieved without direct participation of the MH. Even if the MH permanently fails, state information left on MSSs can be identified and discarded.

2 The Mobile IP Environment

The mobile computing environment used in this paper is based on the Mobile IP architecture [2]. This environment, as illustrated in Figure 1, contains fixed hosts connected by a backbone network and mobile hosts that use a wireless interface to communicate with fixed hosts and other mobile hosts. Each MH is associated with a home network on which the MH receives packets like a normal fixed host. Each MH is also assigned a home address that has the subnet prefix of its home network. The home address never changes, regardless of the MH's movement. Mobile support stations (foreign agents) are those fixed hosts that have both a wireless interface and a fixed network (Ethernet, ATM, etc.) interface. They function as routers and provide connections for mobile hosts to the entire network. The area that a mobile support station's wireless interface serves is called a cell. As mobile hosts move from cell to cell, their IP addresses have to be changed to reflect the subnet mask of their new mobile support stations. This can cause difficulty in maintaining a connection as the MH traverses cells. Mobile IP solves this problem by providing both home agents and foreign agents.

When communicating with a mobile host, other hosts always send packets to the mobile host's home address. A home agent executes on the mobile host's home network. It maintains current location information of the mobile hosts. Packets destined for mobile hosts are intercepted by the home agent and then tunneled to the current foreign agent that is serving the mobile host. Packets are then delivered to the mobile host by the foreign agent. Packets sent by mobile hosts are generally delivered to their destination using standard IP routing mechanisms, not necessarily through the home agent. An example is illustrated in Figure 1. Packets from mobile host A follow the dotted line and pass an FA, a HA, and an FA, before reaching mobile host B.

One of the distinctive features of this architecture is the existence of home agents. When an MH switches cells, the home agent must know where the mobile host is located before any future packets can be delivered to the mobile host. This suggests that the home agent might be an attractive place to log messages for mobile hosts. However there are routing optimizations proposed in the literature [21] that route some packets directly to the foreign agent of the mobile host instead of through its home agent. On the other hand, we observe that all packets sent to mobile hosts must reach the mobile support station first before transmission through the wireless interface. In our approach stable storage at the MSS is used to store checkpoints and message logs for mobile hosts.

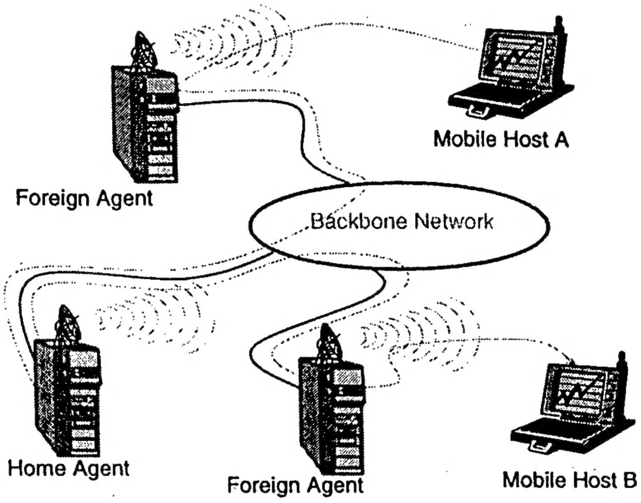

Figure 1: The mobile IP environment.

We assume that the foreign agents and home agents do not fail when serving mobile hosts that are executing the checkpoint and roll-back recovery protocol. MSS and HA typically have a much smaller failure rate than that of MHs as they run on fixed networks. Even if they do fail, since HA and FA are processes that implement carefully defined state machines, checkpointing and message logging protocols can be designed relatively easily to tolerate those failures [22].

There can be multiple processes running on a single mobile host. They can have fail-stop failures independent of each other, or fail at the same time as the mobile host. Finally, we assume that MHs communicate with MSSs using a FIFO link level protocol, processes communicate with each other using TCP (or other reliable transport protocol) over Mobile IP, and processes execute according to the piece wise deterministic model.

3 Related Work

Elnozahy, Johnson, and Wang developed a general survey for checkpointing and message logging protocols in distributed systems [23]. Alvisi and Marzullo have provided an in-depth treatment of message logging [24]. Rao and Alvisi compared the cost of recovery for different message logging approaches [4], and Neves and Fuchs [25] compared recovery speed for a coordinated checkpoint protocol [13] and a sender based message logging protocol [8].

Acharya and Badrinath [15] introduced a two-phase method for taking global consistent checkpoints. They proposed that checkpoints be stored on the stable storage of mobile support stations instead of on mobile hosts. In their protocol, processes alternate between two states, SEND and RECV. If a process is in the SEND mode and receives a

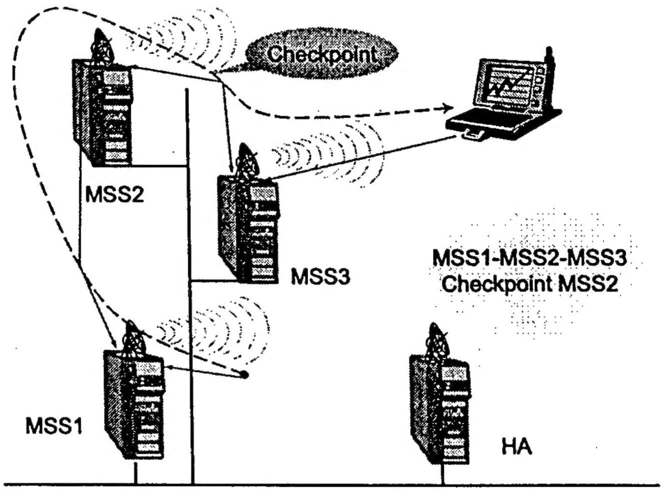

Figure 2: Message logging algorithm.

message, it is forced to take a checkpoint. During recovery, the global state is reconstructed from a set of checkpoints for each process.

Pradhan, et al. analytically evaluated the performance of different state saving protocols and hand off strategies [14]. They also suggested storing checkpoints and message logs at mobile support stations. Their result indicates that message logging is suitable for mobile environments except in cases where the mobile host has high mobility, wireless bandwidth is low, and failure rate is high.

A hybrid checkpoint-recovery protocol for mobile systems was proposed by Higake and Takizawa [17]. As a mobile host moves between cells, it leaves an agent process on each mobile support station on its itinerary. During recovery, processes on fixed hosts recover from consistent checkpoints and processes on mobile hosts restart from their own checkpoints and roll to a state that is consistent with those on fixed hosts with the help of agent processes.

Cao and Singhal proved that it is not possible for a checkpoint algorithm to preserve both non blocking and min process properties at the same time [18]. This work is based on an earlier work that tried to achieve non blocking and min processes at the same time [16]. They also proposed a non-blocking mobile checkpointing protocol [19]. Their scheme uses mutable checkpoints to avoid storing unnecessary checkpoints on mobile support stations.

Neves and Fuchs [13] developed an adaptive checkpointing scheme for mobile computing. Their protocol uses time to indirectly coordinate the creation of recoverable consistent checkpoints. Processes take hard checkpoints that are sent to home agents and soft checkpoints that are stored on the local disk of the mobile host.

4 Algorithm Description

4.1 Message Logging and Checkpointing

The message logging and checkpointing algorithm of this paper consists of three parts: one that executes on the MSS, one that executes at the HA, and one implemented by application processes on the mobile host. The protocol implemented by application processes on the mobile host is similar to existing message logging protocols. Processes tag every sent message and store a copy of the tag in memory. Upon receiving a message, a process also stores the tag of the received message in memory. The tag includes the message sequence number, a globally unique process identifier, and the tag of the last message received by the sending process. The process periodically writes checkpoints to the MSS that is currently serving as its foreign agent. Checkpoints are taken in a non-blocking fashion using copy on write [26]. The checkpoint includes not only the process state information necessary to recover the process, but also the tags of the last message it sent and received before the checkpoint.

Mobile IP maintains the TCP connection as the MH moves, thus switching cells when writing checkpoints does not create problems. Packets sent by the MH are routed as conventional IP, instead of having to go through the HA, thus not degrading performance on the MSS. When the MH switches cells, application processes are notified. Each process sends to the new MSS a *Report Message* that contains the tag of the last message it received from the old MSS. These messages can be piggybacked in the registration packet specified by the Mobile IP protocol.

Every time an MH enters a MSS's cell, the MSS assigns a unique id for the message log of each application process executing the message logging protocol on the MH. If the MH has visited this MSS before, the ids assigned are greater than those previous. Messages destined for mobile hosts in the MSS's cell are logged on the MSS before being forwarded. The sequence of messages seen by the mobile host are the same as seen and logged by the MSS due to the FIFO link-level wireless protocol. Some messages can be logged by the MSS and not yet delivered to the MH. This type of message is an *in transit message*. The new MSS forwards report messages sent to it from the MH at cell switch time to the old MSS of the MH. The old MSS can then use the last received message tag for each application process to detect in transit messages and can purge them from message logs. After each mobile host leaves a cell, the MSS that the MH just left sends a message to the MH's HA reporting the messages log ids for each application process on this MH. If an MH writes a checkpoint to an MSS, the MSS sends a *checkpoint message* to the MH's HA indicating that it has finished storing the checkpoint for the MH. In the event that a process failed on an MH, or the

MH failed, the MSS that was serving the MH detects the error and sends its home agent a *Failed Message* that contains message tags of the last sent message for each failed application to the MH's HA.

On the HA, a process keeps track of the location of an MH using the registration procedure specified by the Mobile IP protocol. It saves the whole itinerary taken by the mobile host into an array named *itinerary array.* Each element of the array is the address of an MSS on the MH's path. The HA waits for the checkpoint message from the MSS that contains the location of the last finished checkpoint of the mobile host. Upon receiving this message, the HA may begin the garbage collection procedure to reclaim stable storage on the MSSs.

The HA also serves as an MSS on the home network. It executes the protocol for MSSs in addition to HA protocols. If an MH does not use the wireless interface on its home network, its HA has to deviate from standard Mobile IP so that the HA can intercept and log messages destined for MHs that are "at home".

In our protocol, checkpoints are taken periodically and stored on MSSs. As one reviewer pointed out, using two kinds of checkpoints [13], one stored on the MH host's local disk and one on the MSSs can reduce power consumption of the mobile host even further (as a smaller number of checkpoints are sent through the wireless interface) and achieve even faster recovery if the local disk survives failure. The HA must be aware of these "possibly stable" checkpoints so that it can know which message logs should be sent to the MH. The associated cost is that larger message logs have to be stored at MSSs and extra processing performed at HAs. Also, if the local disk on the MH fails, recovery can take longer. Depending on the application, network characteristics and type of mobile hosts, this can serve as a viable alternative to saving all checkpoints on MSSs.

If processes running on mobile hosts maintain multiple TCP connections with other processes, they can see a different sequence of messages from those by the foreign agents due to scheduling in the thread library. During recovery they could take an execution path different from that before the failure. To ensure correctness, we limit applications running on the mobile hosts to be a single message queue shared by multiple processes, or a single process with one TCP connection.

Figure 2 illustrates the effects of the logging algorithm. As a mobile host moves along the dashed line, MSSs on its itinerary log messages for the mobile host. The mobile host takes a checkpoint while at MSS2. All this information is sent to the HA for further processing.

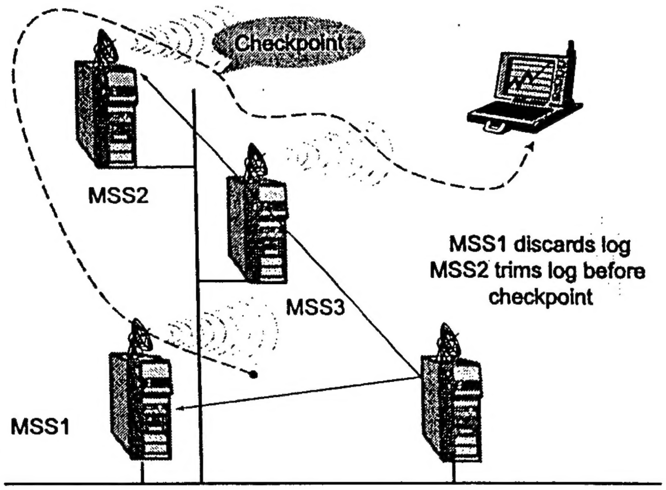

Figure 3: Garbage collection.

4.2 Distributed State Information and Garbage Collection

As the MH moves from cell to cell, it leaves message logs and checkpoints on the MSSs that are on its itinerary. Itinerary information kept at its HA is used to reconstruct a consistent state for the MH. The HA knows on which MSS the last checkpoints are stored for each application process on the MH by examining the checkpoint messages sent to it by MSSs. Messages logged after the checkpoint will have to be replayed in order for processes on the MH to recover. All other logged messages and previously stored checkpoints are no longer needed for recovery.

An example is sketched in Figure 3. Assume there is only one application process P executing on the MH. The MH moves from MSS1 to MSS3 and P takes a checkpoint on MSS2. HA has the MH's itinerary and the location of P's checkpoint stored in its memory. Since the HA knows that a checkpoint for P has been taken at MSS2, the HA can now ask MSS1 to delete message logs for process P, and ask MSS2 to delete messages logged prior to the checkpoint since they are no longer necessary for recovery.

Garbage collection is straight forward. After receiving a checkpoint message from an MSS, the HA examines its itinerary array and determines which message log is useful and which is rendered obsolete by the checkpoint. The HA sends out requests to those MSS with stale message logs and checkpoints so that they can garbage collect the stable storage. This request includes message log identifiers so that the MSS can distinguish among multiple logs in case the MH visits the MSS several times. Upon receiving the request, the MSS removes obsolete message logs and/or checkpoints, then sends back an ACK to HA. After receiving all the ACKs, the HA trims its itinerary array. The HA can also start the garbage collection either periodically to

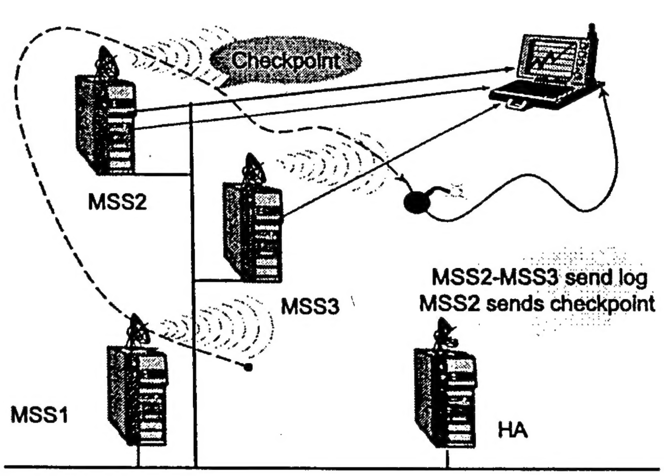

Figure 4: Recovery algorithm.

limit the length of the itinerary array or if stable storage at the MSS becomes depleted.

4.3 Recovery Algorithm

A mobile host restarts a failed process by sending to its HA a message containing the id of the process to be restarted. The HA responds by sending to the MH the message tag of the last message sent out by the process. The HA determines which MSS currently holds the latest checkpoint for this process and asks the MSS to send the checkpoint to the MH. Then the HA sends requests to MSSs that hold message logs for the process, which then in turn replay the log so that the process receives messages in the same order as before failure. When replaying the logged messages, the MSSs mark them as "replayed" so that they are not logged by the receiving MSS. If other processes continue to send messages to the recovering process, these messages will be logged and sent to the MH as normal messages, but they are not delivered to the application until after the recovery is complete. Figure 4 illustrates this procedure.

If an application attempts to send messages during recovery, the message tag is compared to the tag of the last message sent by the process before failure. If the tag indicates that the message has been sent before the failure, the message is not transmitted by the MH. This prevents the MH from re-transmitting application messages previously sent during recovery, thereby saving bandwidth and battery power. Failures during recovery are handled in the same way as failures during normal execution, since messages sent to the MH during recovery are logged on the MSS as normal messages.

4.4 Limited Stable Storage on Mobile Support Stations

If a mobile support station depletes its stable storage while trying to store checkpoints or message logs on behalf of mobile hosts, it has to either halt and perform garbage collection or find alternative storage. Halting an MSS effectively blocks every process on every mobile host in the MSS's cell. All incoming packets for mobile hosts are lost and must be resent later. Managing stable storage on the MSSs to reduce the frequency of blocking is therefore critical. Stable storage management is the focus of another recent project [20].

One way to reduce the possibility of halting an MSS is to use watermarks. When free stable storage on an MSS reaches a low watermark, that MSS selects a process as the target of garbage collection, forces it to take a checkpoint (maybe on another MSS), and discards previous message logs and checkpoints saved for that process.

Halting an MSS can also be avoided when storage is depleted by forwarding the logs and checkpoints to the mobile host's home agent, if there is enough bandwidth in the backbone network. The MSS can then execute the garbage collection algorithm. Another alternative is to have other MSSs or routers on the route of the packets store message logs. The MSS on the last hop can simply forward packets to the MH without logging them. This requires some signaling messages be sent to the HA so that the HA knows the exact locations of the logs.

5 Experimental Results

We compare the performance of our protocol with an ideal coordinated checkpoint protocol that takes periodic checkpoints without exchanging any messages. Failure free overhead and recovery time are evaluated.

5.1 Experiment

The specific environment used in the experiment is shown in Figure 5. A Sun Sparc 20 workstation running Solaris 2.6 with 320M memory and a Lucent Technology Wavepoint II [27] connected by 10M Ethernet served as the mobile support station. Checkpoints and message logs were stored on a dedicated file server that was connected to the workstation using a high speed ATM network. Two Pentium II 300MHz PCs equipped with Lucent Technology's Wavelan [27] wireless interface cards served as the mobile hosts. The PCs were running Windows NT 4.0 with 256M memory each. In our implementation, processes executed pre-generated traces that emulated WWW browsing behavior. The processes also functioned as servers and they read requests and generated replies in both sleep and request states. There were four client processes running, two on each PC. Each client was assigned a unique ID.

Table 1: Execution Time Overhead Comparison.

Trace	unmodified (seconds)	ckpt (seconds)	ckpt Overhead(%)	log with ckpt (seconds)	log with ckpt Overhead(%)
T1	1180.2	1243.6	5.3	1234.5	4.60
T2	869.1	9067.5	4.3	915.5	5.34
T3	1013.4	1060.1	4.6	1066.1	5.20
T4	871.5	898.1	3.0	905.6	3.92
T5	911.3	957.6	5.0	950.6	4.32
T6	866.7	910.0	4.9	913.1	5.35
T7	850.7	891.9	4.8	876.6	3.04

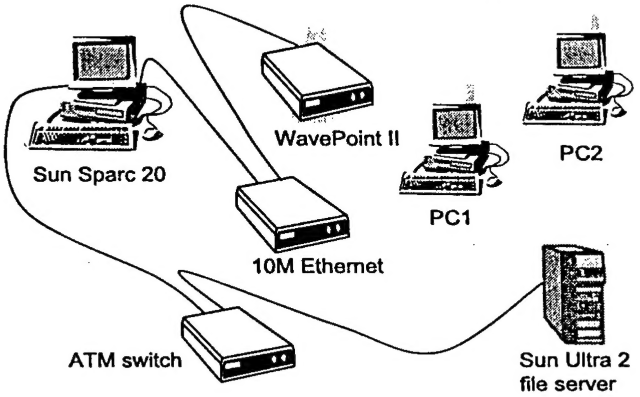

Figure 5: Experiment environment.

For routing and processing, a client attached to each message it transmitted a header that contained the destination ID, the type of message (request or reply), and the ID of the sender. Clients did not send messages directly to each other. Instead, they sent messages over the wireless network to a server process running on the Sun workstation. The server process was responsible for storing checkpoints and message logs as requested by the client processes. It also functioned as a router by examining the header field of each message and forwarding messages to their destination.

Checkpoints were not actual restartable process states, but were large messages intended to represent a range of reasonable state sizes for mobile hosts. Client processes had a separate thread that periodically sent checkpoints to a server. The server process also had separate threads that listened for checkpoints and wrote them to disk. Checkpointing was asynchronous.

5.2 Trace Generation

Request traces were generated from four individual traces to represent a variety of network load. These four individual traces represented the proc̃ess sleep interval (SLEEP), number of requests sent during each active interval (NUMBER), the request packet distribution (RQ), and the reply packet distribution (RY). The request packet distribution was small as HTTP requests are typically less than several hundred bytes. Reply packets were large with a large variance to reflect the nature of a typical webservers' output. When accessing a web page, several requests are typically sent to the web server. We captured this behavior with the NUMBER trace.

A utility program read the first three traces and generated request traces used by each client. This program first read a value from the SLEEP trace and multiplied it by 1000 and a predefined coefficient to obtain the sleep time in milliseconds. The time value was written into the request trace file. The NUMBER trace was then read to determine how many request events were to be written out and the destination for these requests was randomly chosen. For every request event, the request length was read from the RQ trace. Seven traces (T1 ... T7) were generated, with coefficient of 1/2, 1/4, ... 1/128. By using several ratios, we obtained a variety of traces that represent distinct network loads.

5.3 Failure Free Overhead

The applications were first executed unmodified without message logging and checkpointing. They then ran with only periodic checkpointing enabled. Finally, the application executed with both checkpointing and message logging enabled. The checkpointing interval was five minutes. The execution times shown in the following figures and tables are the average of three runs. Total execution time of the three cases for different traces was measured and is shown in Table 1 and Figure 6. The overheads in-

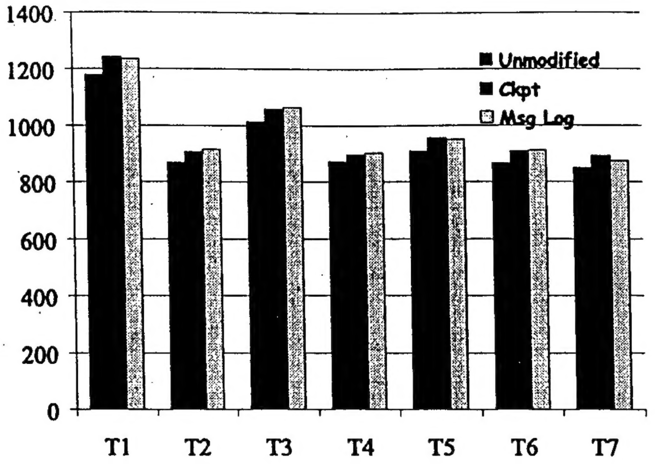

Figure 6: Normal execution overhead.

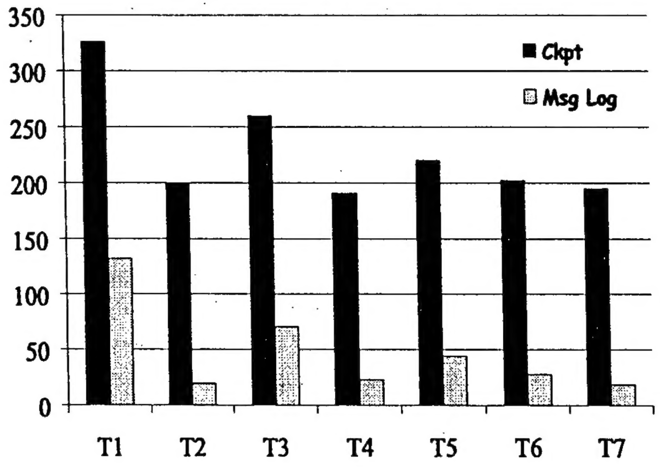

Figure 7: Recovery time.

curred by the coordinated checkpointing protocol and the message logging protocol are also shown.

5.4 Recovery Time

Recovery time is obtained by measuring the time for a process to read the checkpoint and proceed to a specific execution point. In our experiments, we chose that point to be after the 500th event in the request trace file. No checkpointing or message logging takes place during recovery. Recovery times for the consistent checkpointing protocol and the message logging protocol are measured and shown in Table 2 and Figure 7.

Table 2: Recovery Time Comparison.

Trace	Using ckpts (seconds)	Using message logs (seconds)
T1	326.5	131.9
T2	200.4	19.82
T3	260.0	70.28
T4	191.00	23.16
T5	220.9	44.14
T6	202.6	27.67
T7	195.5	18.98

5.5 Discussion

From the failure free overhead data we see that for all of the traces message logging has similar performance to checkpointing without logging. The largest difference is less than 2 percent of the execution time without any checkpoints. When message logging is performed, the processes take checkpoints at the same frequency as the standard checkpointing protocol. The experiments illustrate that the overhead due to logging messages at the MSS is negligible. The reason is that messages are not written to disk before being forwarded to the MH.

As is known for fixed networks, recovery using message logging is often faster than that based on standard checkpoints. The processes recovered three to ten times faster using message logs than with checkpoints in our experiments with the Wavelan wireless network. Processes did not have to block and wait for other processes to transmit messages. Messages were also transmitted over the wireless link just once instead of twice, as in normal execution, resulting in less contention on the wireless network and lower latency for message transmission.

An interesting phenomenon is that for some traces the overhead due to message logging and checkpointing is actually less than that due to checkpointing. One explanation is that the action of logging message changed the timing of messages transmitted on the wireless network and thereby contentions were reduced.

6 Conclusions

This paper described a message logging protocol for mobile hosts, mobile support stations and home agents in a Mobile IP environment. An approach to organizing the distributed state information was also presented. The organizing scheme provides easy garbage collection without participation from mobile hosts and can tolerate the case where some mobile support stations do not have enough stable storage for mobile hosts to store state information.

Failure free overhead and recovery speed were compared between an ideal consistent checkpoint protocol and our message logging protocol. Message logging incurred only marginally larger overhead during failure free operation compared to the ideal consistent checkpointing protocol. Message logging has a decided advantage in recovery with mobile wireless networks.

Acknowledgment

We take this opportunity to thank the anonymous referees for their comments.

References

[1] C. Perkins, *Mobile IP Design Principles and Practices*. Addison-Wesley, 1997.

[2] C. P. (ed.), "IPv4 Mobility Support," *RFC 2002*, October 1996.

[3] B. R. Badrinath, A. Acharya, and T. Imielinski, "Impact of Mobility on Distributed Computations," *SIGOPS Review*, pp. 15–20, April 1993.

[4] L. A. S. Rao and H. Vin, "The Cost of Recovery in Message Logging Protocols," *Proceedings of the 17th Symposium on Reliable Distributed Systems*, pp. 10–18, October 1998.

[5] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Transactions on Computer Systems*, vol. 3, no. 1, pp. 63–75, February 1985.

[6] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 1, pp. 23–31, January 1987.

[7] R. E. Strom and S. Yemini, "Optimistic recovery in distributed systems," *ACM Transactions on Computer Systems*, vol. 3, no. 3, pp. 204–226, August 1985.

[8] D. B. Johnson and W. Zwaenepoel, "Sender-based Message Logging," *Proceedings of the 17th International Symposium on Fault-Tolerant Computing*, pp. 14–19, July 1987.

[9] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "The Performance of Consistent Checkpointing," *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pp. 39–47, October 1992.

[10] L. M. Silva and J. G. Silva, "Global Checkpointing for Distributed Programs," *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pp. 155–162, October 1992.

[11] J. Plank and K. Li, "Faster Checkpointing with N+1 Parity," *Proceedings of the 24th International Symposium on Fault-Tolerant Computing*, pp. 288–297, August 1994.

[12] Y.-M. Wang, Y. Huang, P. Vo, P.-Y. Chung, and C. Kintala, "Checkpointing and its Applications," *Proceedings of the 25th International Symposium on Fault-Tolerant Computing*, pp. 22–31, June 1995.

[13] N. Neves and W. K. Fuchs, "Adaptive Recovery for Mobile Environments," *Communications of the ACM*, vol. 40, no. 1, pp. 68–74, January 1997.

[14] D. K. Pradhan, P. Krishna, and N. H. Vaidya, "Recovery in Mobile Environments: Design and Trade-off Analysis," *Proceedings of the 26th International Symposium on Fault-Tolerant Computing*, pp. 16–25, June 1996.

[15] A. Acharya and B. Badrinath, "Checkpointing Distributed Applications on Mobile Computers," *Proceedings of the 3rd International Conference on Parallel and Distributed Information systems*, pp. 73–80, September 1994.

[16] R. Prakash and M. Singhal, "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1035–1048, October 1996.

[17] H. Higaki and M. Takizawa, "Checkpoint-Recovery Protocol for Reliable Mobile Systems," *Proceedings of the 17th Symposium on Reliable Distributed Systems*, pp. 93–99, October 1998.

[18] G. Cao and M. Singhal, "On the Impossibility of Min-Process Non-Blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems," *Proceeding of the 27th International Conference on Parallel Processing*, pp. 37–44, August 1998.

[19] G. Cao and M. Singhal, "Low-Cost Checkpointing with Mutable Checkpoints in Mobile Computing Systems," *Proceedings of the 18th International Conference on Distributed Computing System*, pp. 462–471, May 1998.

[20] K. Ssu, W.K. Fuchs, and N. Neves, "Adaptive Checkpointing with Storage Management for Mobile Environments," *manuscript*, December 1998.

[21] D. B. Johnson, "Scalable and Robust Internetwork Routing for Mobile Hosts," *Proceedings of the 14th International Conference on Distributed Computing Systems*, pp. 2–11, June 1994.

[22] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys*, vol. 22, no. 4, pp. 299–319, December 1990.

[23] E. Elnozahy, D. Johnson, and Y.-M. Wang, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," Tech. Rep. CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, October 1996.

[24] L. Alvisi and K. Marzullo, "Message Logging: Pessimistic, Optmistic, Causal, and Optimal," *IEEE Transactions on Software Engineering*, pp. 149–159, February 1998.

[25] N. Neves and W. K. Fuchs, "RENEW: A Tool for Fast and Efficient Implementation of Checkpoint Protocols," *Proceedings of the 28th International Symposium on Fault-Tolerant Computing*, pp. 58–67, June 1998.

[26] J. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent Checkpointing under Unix," *Usenix Winter 1995 Technical Conference*, pp. 213–233, January 1995.

[27] *http://www.wavelan.com.*